Table of contents

Introduction

This report provides a concise summary of the task assigned to me by Sereact, where I worked on aligning the LLaVA-1.6 7B Vision-Language Model (VLM) with human preferences using the RLAIF-V dataset. The alignment was achieved using the ORPO (Monolithic Preference Optimization without Reference Model) method. The report outlines my findings, insights gained during the process, and suggestions for future improvements. Additionally, it includes information on accessing the code for training and inference with the trained model.

Approach and challenges

I had previous experience with vision-language models through my work on coding the "Pali-gemma model from scratch," which gave me a strong understanding of how vision-language models function and what happens inside them. This foundational knowledge was crucial for grasping what the current VLM, the "LLaVA-1.6 7B VLM," expects and how it operates. However, I had no prior experience with fine-tuning large language models (LLMs). To address this, I conducted in-depth research on various topics, including LLM fine-tuning, instruction fine-tuning, alignment, and methods such as RLHF and RLAIF. I also reviewed the LLAVA research paper in great detail. The github repo contains my personal notes that I have made from all my research from the internet.

The core concept is that we typically start with a base model, which is initially trained on a vast amount of publicly available data from the internet. This forms a general-purpose model, which can then be fine-tuned for specific tasks. For example, GPT-4 is a base model trained on extensive internet data. However, on its own, it does not excel in following instructions or engaging in conversations effectively. After it underwent instruction fine-tuning, the result was ChatGPT, which is more capable of interacting with users in a conversational manner.

Once models are fine-tuned for a specific task, we can further align them to human preferences to enhance their real-world utility and safety. This alignment process ensures that the model's outputs are more natural, helpful, and less prone to generating harmful, biased, or incorrect content. One way to achieve this is by using a reward model trained with human feedback to evaluate and guide the model's responses toward human-aligned outputs. Alternatively, the model itself can be used to assess and refine its own answers to better align with human preferences. This approach is central to ORPO (Monolithic Preference Optimization without Reference Model), where we align models without relying on a separate reference model to determine human-aligned responses.

Since LLaVA is already an instruction fine-tuned model, our goal is to take the pretrained version of the model and then perform alignment tasks on the model such that the responses provided by the model are more natural and helpful, and less harmful, biased or incorrect.

Becoming one with the data

The dataset used for this task is the [RLAIF-V dataset](#) , which contains images, questions related to each image, a chosen response that aligns well with human preferences, and a rejected response that does not. By training the model on this dataset using ORPO, the goal is to guide the model to produce responses that resemble the chosen ones and avoid generating responses like the rejected ones.

Since this is a Hugging Face dataset, I loaded it and removed any unnecessary columns. Due to resource constraints, I experimented with different data sizes for training. Ultimately, my final model was trained on 5% of the total dataset, which comprises 4,157 samples. I then split the dataset into 85% for training and 15% for validation.

```
[ ] ds

    Dataset({
        features: ['image', 'question', 'chosen', 'rejected'],
        num_rows: 4157
    })

[ ] train_test_split_datasets = ds.train_test_split(test_size=0.15)
    train_test_split_datasets["train"]

    Dataset({
        features: ['image', 'question', 'chosen', 'rejected'],
        num_rows: 3533
    })

[ ] train_test_split_datasets["test"]

    Dataset({
        features: ['image', 'question', 'chosen', 'rejected'],
        num_rows: 624
    })
```

Figure description: Dataset, training set and validation set

Data collators

Data collators are responsible for presenting the training and validation sets to the model in the format the model and corresponding trainer APIs are expecting. This was one of the most challenging parts of the task as the documentation regarding the default format for ORPO Trainer APIs and a vision language model such as LLaVA is not clear in the official documentation. I could not find ORPOTrainer API being used with a vision language model being used anywhere in publicly available projects on GitHub. This presents a problem, as normally ORPOTrainer API has prompts, chosen and rejected columns in the dataset. It was challenging to make such a trainer work with a multimodal

model with both text and images as input. As it was not clear with ORPOTrainer API, how can I pass multimodal inputs to the model, I have coded a custom data collator function to pass in when I am instantiating the Trainer API.

```python
prompts = []
images = []
chosen_responses = []
rejected_responses = []

# Process each entry in the batch
for example in examples:
    image = example["image"]  # Get the image (used directly in the prompt)
    question = example["question"]  # Get the question
    chosen = example["chosen"]  # Get the chosen response
    rejected = example["rejected"]  # Get the rejected response

    # Construct the prompt in the model's expected format using a raw string

    prompt = f"[INST] <image>\n Provide answer to the following question that best aligns with human preferences. {question} [/INST]"

    # Append to respective lists
    images.append(image)
    prompts.append(prompt)
    chosen_responses.append(chosen)
    rejected_responses.append(rejected)

# Process the prompt and image for the input batch
batch = self.processor(text=prompts, images=images, padding=True, truncation=True, max_length=MAX_LENGTH, return_tensors="pt")
batch = batch.to(torch.float16)

# Tokenize chosen and rejected responses
chosen_batch = self.processor.tokenizer(text=chosen_responses, padding=True, truncation=True, max_length=MAX_LENGTH, return_tensors="pt")
rejected_batch = self.processor.tokenizer(text=rejected_responses, padding=True, truncation=True, max_length=MAX_LENGTH, return_tensors="pt")

batch["chosen_input_ids"] = chosen_batch["input_ids"]
batch["chosen_attention_mask"] = chosen_batch["attention_mask"]
batch["rejected_input_ids"] = rejected_batch["input_ids"]
batch["rejected_attention_mask"] = rejected_batch["attention_mask"]

# Create labels for chosen and rejected responses, masking padding tokens
chosen_labels = chosen_batch["input_ids"].clone()
rejected_labels = rejected_batch["input_ids"].clone()

# Mask padding tokens with -100 to ignore them in the loss function
chosen_labels[chosen_labels == self.processor.tokenizer.pad_token_id] = -100
rejected_labels[rejected_labels == self.processor.tokenizer.pad_token_id] = -100

# Add labels to the batch
batch["chosen_labels"] = chosen_labels
batch["rejected_labels"] = rejected_labels

# labels = batch["input_ids"].clone()
# labels[labels == processor.tokenizer.pad_token_id] = -100
# batch["labels"] = labels

return batch
```

Figure description: Custom data collator

I have performed a few experiments with this data collator block. Where I am restructuring the prompts, questions, chosen and rejected responses to the model. But, the model was always overfitting and causing problems when I was experimenting with different sizes of datasets. So, I was unsure if the model was able to see the images and piece everything together successfully in the right format.

So, then I have finally used the approach followed in DPOTrainer when it is being used with multimodal inputs. The function to collate data in my final version of code was using a method to reformat the training and validation sets to a to the required structure that the ORPOTrainer is expecting following the way that DPOTrainer API usually does when handling multimodal data. I have

done this as it is mentioned in the official documentation that ORPOTrainer API expects a similar dataset format of DPOTrainer API and DPOTrainer has documentation on how to deal with multimodal data.

```python
def collate_fn(batch):
    # Initialize lists to hold data
    images= []
    prompts = []
    chosen_responses = []
    rejected_responses = []

    # Process each entry in the batch
    for entry in batch:
      image = entry["image"]  # Get the image (used directly in the prompt)
      question = entry["question"]  # Get the question
      chosen = entry["chosen"]  # Get the chosen response
      rejected = entry["rejected"]  # Get the rejected response

      # Construct the prompt in the model's expected format using a raw string

      prompt = f"[INST] <image>\nProvide answer to the following question that best aligns with human preferences.{question} [/INST]{chosen}"

      # Append to respective lists
      images.append(image)
      prompts.append(prompt)
      chosen_responses.append(chosen)
      rejected_responses.append(rejected)

    # Use Dataset.from_dict() from the 'datasets' library
    orpo_dataset = Dataset.from_dict({
        "images":images,
        "prompt": prompts,
        "chosen": chosen_responses,
        "rejected": rejected_responses,
    })
```

Figure description: final collate function being used

Loading the model

I have used a Q-LoRA based 4-bit quantized model of the "llava-v1.6-mistral-7b" model as I don't have a lot of computing resource available to perform full fine-tuning on the model. **Q-LoRA** updates only a small number of low-rank adapter parameters, instead of the full set of model parameters. Q-LoRA reduces weight matrices into low-rank matrices using matrix decomposition. Instead of updating the full weight matrices during training, Q-LoRA applies low-rank adaptation, which approximates the changes to weights by updating only small, low-rank matrices. Q-LoRA also uses quantization techniques (such as 4-bit quantization) to reduce the precision of the model's weights and activations. By storing weights in lower precision, it reduces the memory footprint, allowing larger models to be fine-tuned using less hardware.

After loading the base model, we will add LoRa adapter layers, which are the only layers we will train while keeping the base model frozen. The key difference with other models lies in the specific layers where these adapters are added, referred to as `target_modules` in PEFT. The choice of these layers can vary depending on the model. In this case, I followed the approach used in the original `find_all_linear_names` function from the LLaVa repository
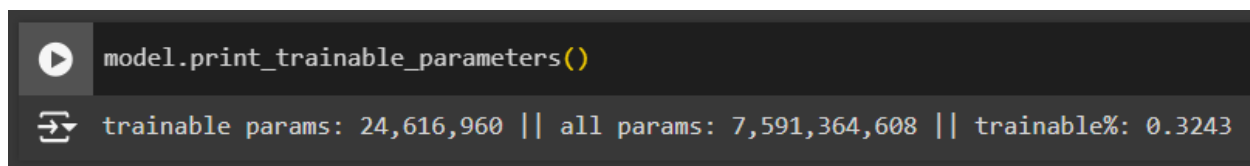
Language model part: It makes sense to add adapters to the language model part, as this is where the primary text generation and response alignment occurs. We are working with human preferences

in text (chosen vs. rejected), so the language model needs to be fine-tuned to prioritize preferred responses.

Vision model: Since our dataset includes images (and your task involves aligning preferences related to image-text interactions), it could also be beneficial to add adapters to the vision model part. This ensures the vision component also gets fine-tuned for better understanding of images in a way that aligns with your preferences.

Multimodal projector: Given that our task involves both images and text, the multimodal projector (which merges image and text representations) plays a key role. Adding adapters here would help improve how the model combines visual and textual information in alignment with the preferred responses.
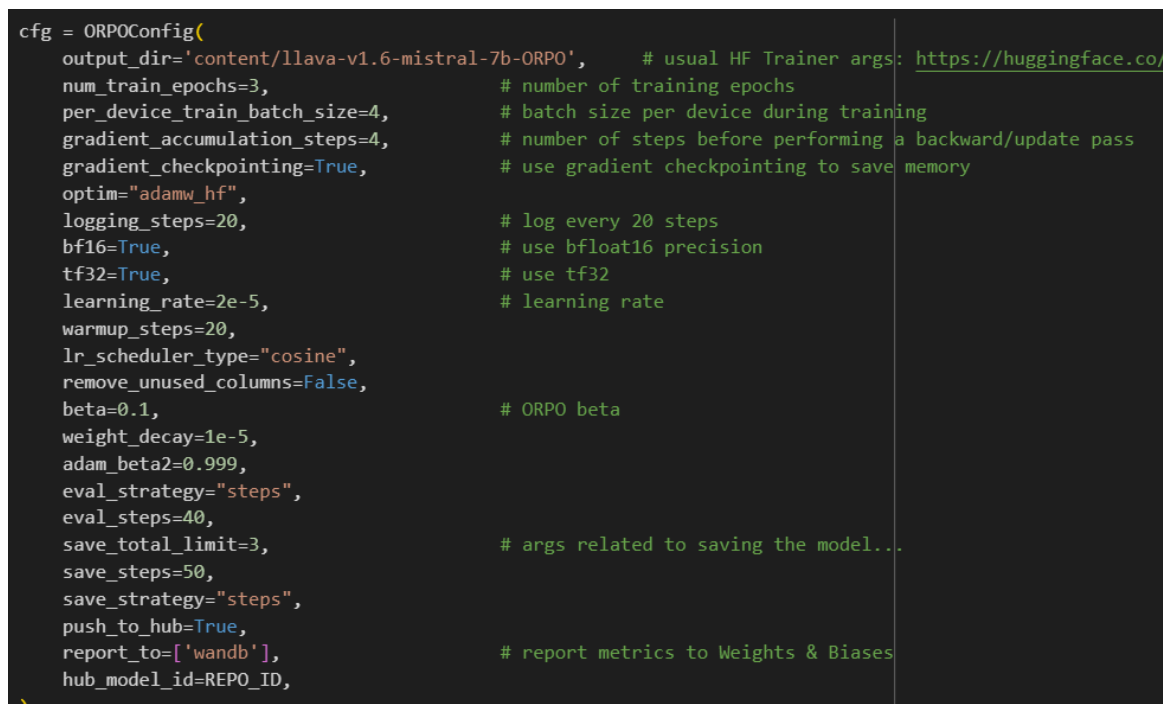
Hence, in our use case, adding adapters to the vision encoder, multimodal projector, and the language model part would likely yield better results.

```
model.print_trainable_parameters()

trainable params: 24,616,960 || all params: 7,591,364,608 || trainable%: 0.3243
```

Figure description : Number of trainable parameters and entire parameters

Hyperparameters:

```python
cfg = ORPOConfig(
    output_dir='content/llava-v1.6-mistral-7b-ORPO',    # usual HF Trainer args: https://huggingface.co/
    num_train_epochs=3,                     # number of training epochs
    per_device_train_batch_size=4,          # batch size per device during training
    gradient_accumulation_steps=4,          # number of steps before performing a backward/update pass
    gradient_checkpointing=True,            # use gradient checkpointing to save memory
    optim="adamw_hf",
    logging_steps=20,                       # log every 20 steps
    bf16=True,                              # use bfloat16 precision
    tf32=True,                              # use tf32
    learning_rate=2e-5,                     # learning rate
    warmup_steps=20,
    lr_scheduler_type="cosine",
    remove_unused_columns=False,
    beta=0.1,                               # ORPO beta
    weight_decay=1e-5,
    adam_beta2=0.999,
    eval_strategy="steps",
    eval_steps=40,
    save_total_limit=3,                     # args related to saving the model...
    save_steps=50,
    save_strategy="steps",
    push_to_hub=True,
    report_to=['wandb'],                    # report metrics to Weights & Biases
    hub_model_id=REPO_ID,
)
```

Figure description : list of Hyperparameters

The above are the hyperparameters used to train the model. I have experimented with various values of batch sizes, gradient accumulation steps, learning rates, weight decays, optimizers, beta values to find the best set of hyperparameters to train the model.

Training

This part of code deals with training the model using ORPOTrainer API by passing the hyperparameters as part of a configuration file. Initially, I was training the model using 1% data from the RLAIF-V dataset which has 841 samples in total. This caused the model to overfit, no matter what the hyperparameter configurations are. Initially, I was training using "T4 GPU" as provided by the google colab and it took hours of training to train the model. So, I have purchased Google colab pro and trained the model using A100 and increased the dataset to 5%. Then, I have performed various experiments to find the right set of hyperparameters to ensure that the model is not overfitting.

Inference

I have saved the model and uploaded the model to the Hugging Face repository. As I have freezed the model and trained the adapter layers added to the relevant modules of the model, only the files related to adapter layers are added to the Hugging Face repository. So, in the "Inference" section, I have loaded the pretrained LLaVA 1.6 mistral 7B model and added my trained adapter layers on top of it again.

Then, I have taken a random sample from the validation set to perform inference on the sample.

Gradio:

In the last part of the "Inference" section, the code to host the model on Gradio is also added. By running the relevant section, we can host the model temporarily in the given link for 72 hours. After launching, you can access the GUI to upload any image and ask any question about the image to receive an answer from the model.

Codebase organization

The entire code to train and make inferences on the model is one single colab notebook. All the installations required are made inside the notebook. The colab notebook, report, my personal notes documents, I have taken down when I was learning about fine-tuning, aligning etc. were all uploaded in a GitHub repo that I have shared.

Launching Training and validation

You can run all the corresponding cells until "Training" to start the training of the model. However, if you want to skip the training and head directly to inference, it is important that you run the cells until the data collator block to create the validation dataset as inference is initially made on one random sample from the validation set.

Inference results

Inferencing using a sample from the validation set

Test image:
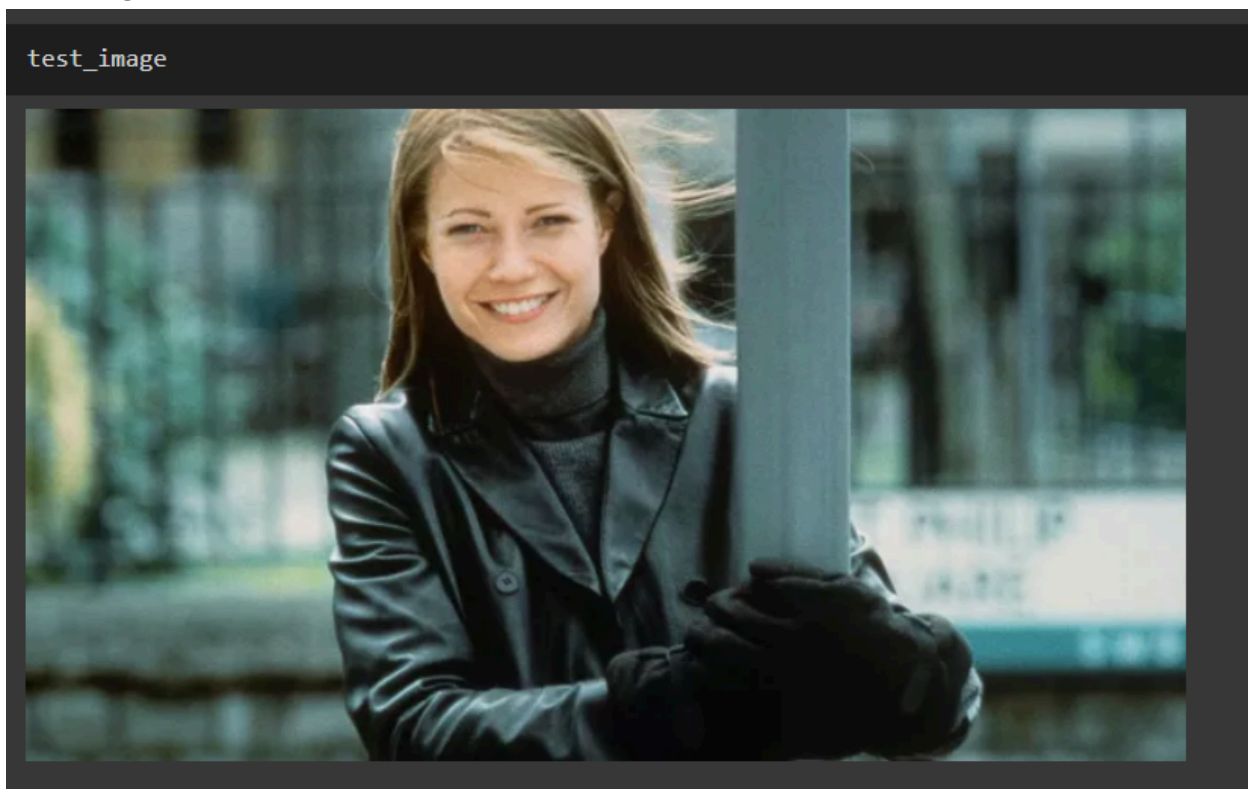
**Question, chosen response and rejected response:**

```
("Look at the image and describe the celebrity's facial expressions, clothing, and any distinctive features.",
 'The celebrity has a smiling face, wears black gloves and a black leather jacket with buttons.',
 'The woman has a smiling face, long brown hair, and is wearing a black coat and gloves.')
```

Response by the model after aligning:

```
[ ]  generated_texts

⇄  'The celebrity, a woman, is smiling and appears to be in a good mood. She is wearing a black leather jacket and has long blonde hair. She is also hold
    ing onto a pole with her hands.'
```

Inferencing using API hosted by gradio
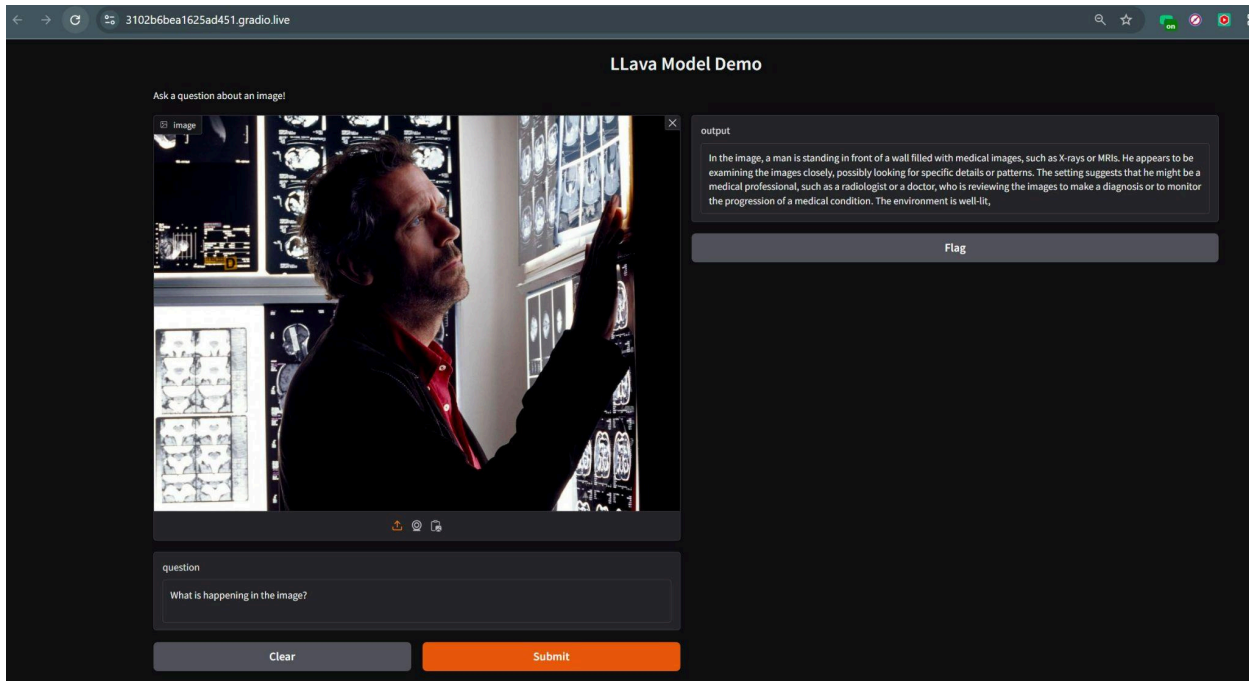
When the code corresponding to hosting the model on Gradio is run, it results in the launch of an interface such as above, where we can upload an image, ask a question and click "submit". The model responds with an answer.

Wandb Results

## Training dashboard plots

1)　Training: loss:

train/loss
— kind-galaxy-11

This is the training loss of the model and is supposed to decrease.

2)　Training: nll_loss (Negative Log-Likelihood Loss):

**train/nll_loss**
— kind-galaxy-11

This is a specific form of loss used in classification tasks, representing how likely the model considers the true labels (chosen responses) based on its predictions. **Desired behavior**: This loss should **decrease**. A lower NLL indicates that the model is increasingly assigning higher probabilities to the correct (chosen) responses.

## 3)   Training: log_odds_ratio:
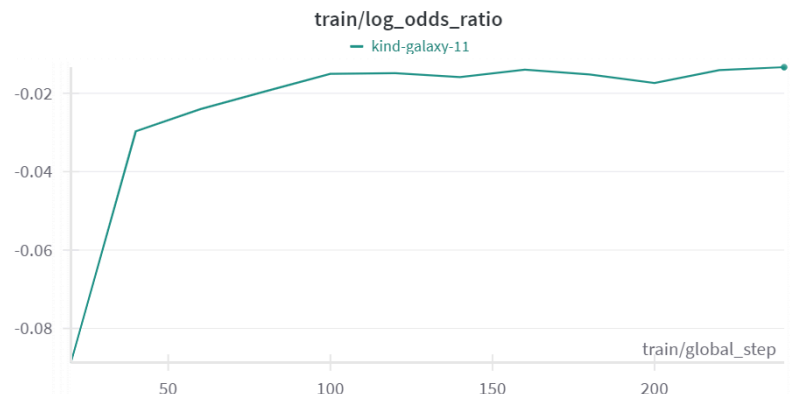
The **log odds ratio** measures the difference in log probabilities between the chosen and rejected responses. **Desired behavior**: This value should **increase** over time.A higher log odds ratio means the model is increasingly more confident in the chosen responses compared to the rejected ones, which is exactly what you want.

## 4)   Training: logps/rejected & logps/chosen:
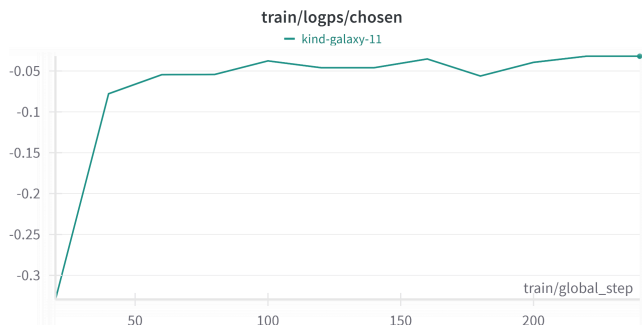
Higher log-probability for chosen responses means the model is more confident in generating those, which is the goal of alignment.

5)   Training: rewards/margins:
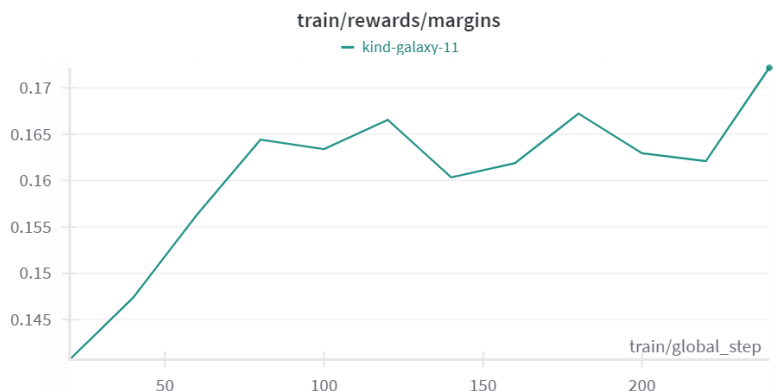
This likely represents the **reward margin**, which is the difference in reward between the chosen and rejected responses. **Desired behavior**: The margin should **increase** over time. A larger margin indicates the model is differentiating more clearly between chosen and rejected responses.

6)   Training: rewards/accuracies:
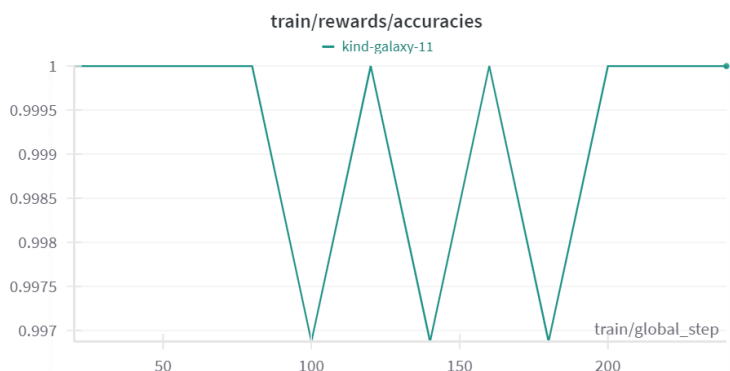
This tracks the model's **accuracy** in predicting the chosen responses over rejected ones.**Desired behavior**: Accuracy should **increase** over time. Higher accuracy means the model is correctly identifying and generating more human-preferred (chosen) responses.

Although the plot looks like this, you can observe that the accuracy is pretty much close to 1 all the time.
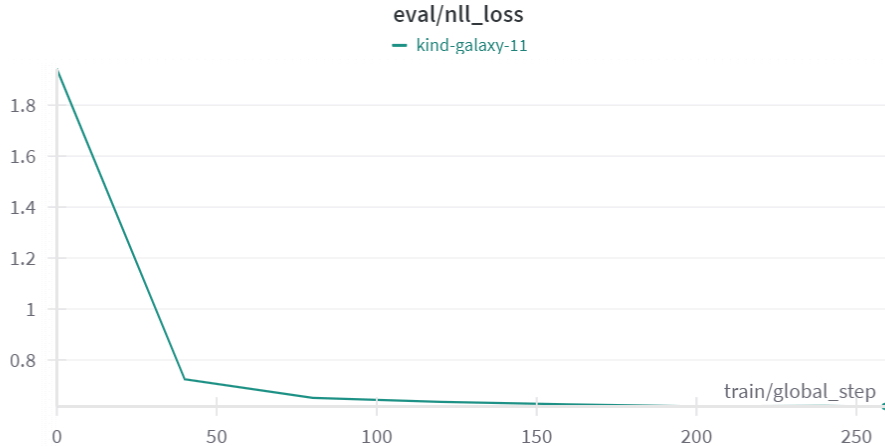
# Evaluation dashboard plots

7)   Evaluation: Loss:
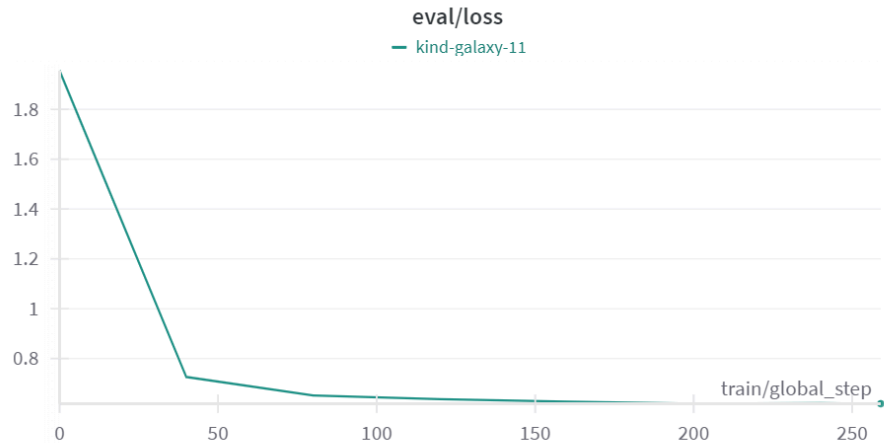
This is the evaluation loss of the model and is decreasing.

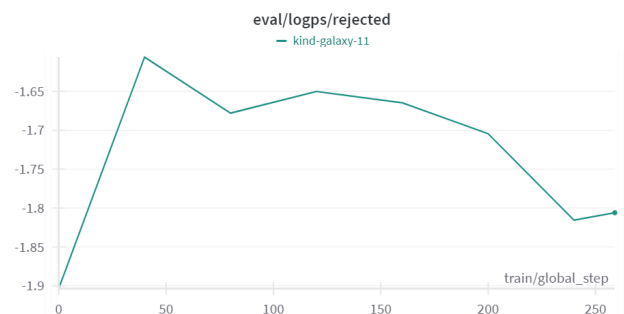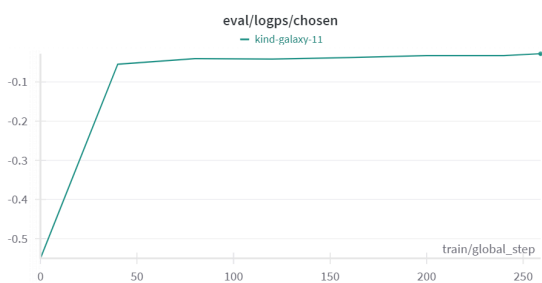## 8)   Evaluation: nll_loss (Negative Log-Likelihood Loss):

This is a specific form of loss used in classification tasks, representing how likely the model considers the true labels (chosen responses) based on its predictions. **Desired behavior**: This loss should **decrease**. A lower NLL indicates that the model is increasingly assigning higher probabilities to the correct (chosen) responses.

9) Evaluation: log_odds_ratio:
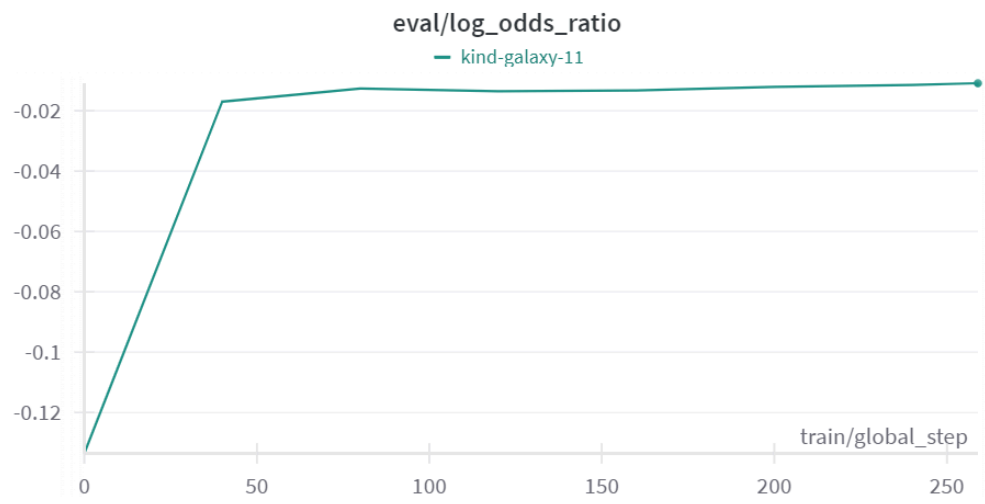
The **log odds ratio** measures the difference in log probabilities between the chosen and rejected responses. **Desired behavior**: This value should **increase** over time. A higher log odds ratio means the model is increasingly more confident in the chosen responses compared to the rejected ones, which is exactly what you want.

10) Evaluation: logps/rejected & logps/chosen:

Higher log-probability for chosen responses means the model is more confident in generating those, which is the goal of alignment.

11) Evaluation: rewards/margins:
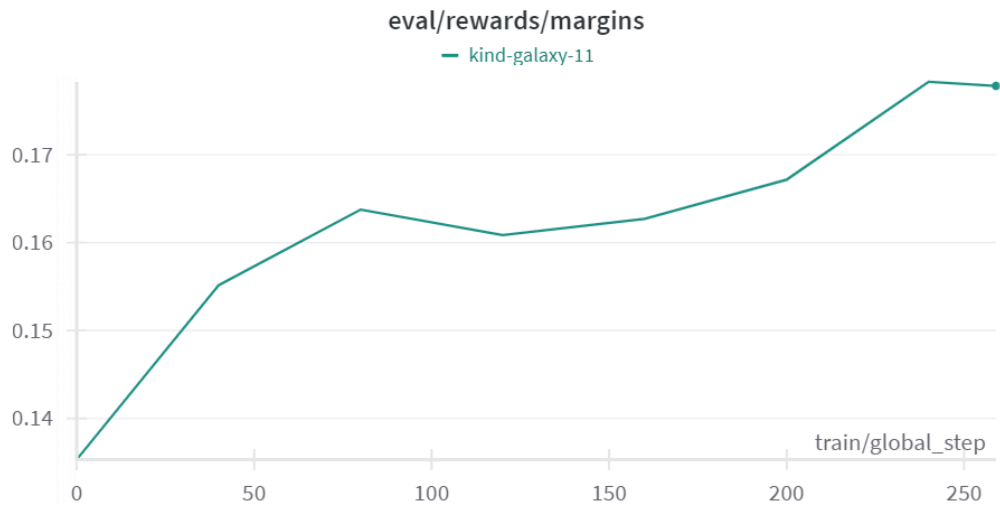
eval/rewards/margins
— kind-galaxy-11

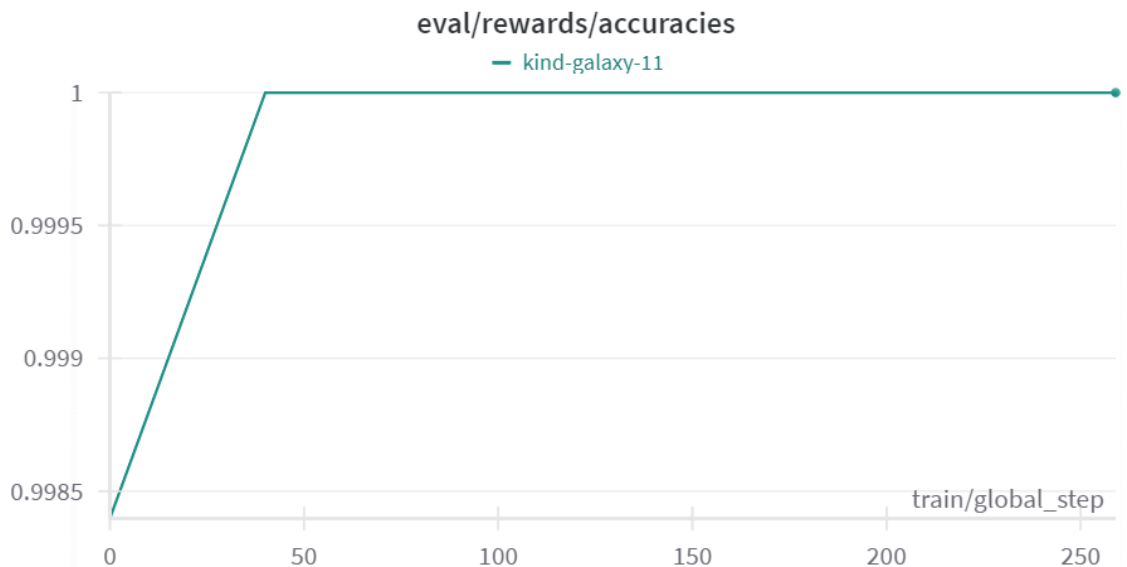This likely represents the **reward margin**, which is the difference in reward between the chosen and rejected responses. **Desired behavior**: The margin should **increase** over time. A larger margin indicates the model is differentiating more clearly between chosen and rejected responses.

12) Evaluation: rewards/accuracies:

eval/rewards/accuracies
— kind-galaxy-11

This tracks the model's **accuracy** in predicting the chosen responses over rejected ones.**Desired behavior**: Accuracy should **increase** over time. Higher accuracy means the model is correctly identifying and generating more human-preferred (chosen) responses.

Results Analysis and Insights

The model after training has a reward accuracy 1. So, the model has learnt a perfectly good way of choosing response over the rejected response. The model might be correctly classifying the chosen responses, but it could still be doing so with less-than-perfect confidence. For example, even though it picks the correct response, it may not assign a probability of exactly 1 to that response and 0 to others. So, the decreasing validation loss and increasing rewards/margins signify that the model is getting better at generating responses close to the chosen response.

Challenges

1) I was initially using 1% of the RLAIF-V dataset, which amounts to 841 samples. So, no matter what combination of hyperparameters I was using, the model is overfitting with a consistent drop in training loss and consistent increase in validation loss. So, I chose to buy more compute and trained using 5% of the RLAIF-V dataset which amounts to 4157 samples.

2) The free tier GPU (T4) was quite slow, requiring me to wait for hours to update hyperparameters and retrain the model. I then upgraded to Google Colab Pro, which was significantly faster, but the available compute resources were still limited, making it challenging to work efficiently.

3) Unclear documentation of how to collate my data when I am using ORPO Trainer. In DPO Trainer, there was a block of information, describing how we are supposed to use it in the case of multimodal models. Such clear instructions were missing in the official documentation of the ORPOTrainer API where the instructions were only about text prompts.

Tasks and bonus tasks progress

1) I have chosen QLoRa with 4-bit quantization to load the model. I have used BitsAndBytes. I have tried using GPTQ to quantize the model. But, GPTQ is not yet supporting the model that we are using in our task i.e., LLaVANext. The following is the error message from GPTQ.

```
WARNING:auto_gptq.nn_modules.qlinear.qlinear_cuda:CUDA extension not installed.
WARNING:auto_gptq.nn_modules.qlinear.qlinear_cuda_old:CUDA extension not installed.
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-97-e08ddfdcc915> in <cell line: 9>()
      7
      8 # Load the GPTQ quantized model
----> 9 model = AutoGPTQForCausalLM.from_quantized(
     10     MODEL_ID,
     11     model_basename="gptq_model.bin",  # Change this to your quantized model's basename

                                     ↕ 1 frames
/usr/local/lib/python3.10/dist-packages/auto_gptq/modeling/_utils.py in check_and_get_model_type(model_dir,
trust_remote_code)
    303     config = AutoConfig.from_pretrained(model_dir, trust_remote_code=trust_remote_code)
    304     if config.model_type not in SUPPORTED_MODELS:
--> 305         raise TypeError(f"{config.model_type} isn't supported yet.")
    306     model_type = config.model_type
    307     return model_type

TypeError: llava_next isn't supported yet.
```

I have also tried using AWQ. But, using it created many conflicts with the libraries from hugging face trl, which we need as we are using ORPOTrainer API from it. So, I have chosen to use BitsAndBytes config instead.

2) Successfully hosted the model on Gradio. I have also tried serving the model to Huggingface TGI and Hugging face spaces. But, the model is too large to fit in within the free tier offerings of the respective websites. It definitely needs GPU to generate responses and both Huggingface TGI and Hugging face spaces are not giving free GPU support.

Future possible improvements/fixes

I am limited by my hardware resourccould not train the model much longer as I don't have enough computational resources. So, with extra compute, the model can be trained with much better hyperparameter configurations with much larger amounts of data, leading to much better results with the model.

Techniques like KL-divergence can be implemented to see if the model is straying away from its original task, while focussing on the alignment tasks.

Conclusion

I approached the challenge using all the resources I had available and done my best to achieve the expected results. I'm truly grateful for the opportunity, as it has allowed me to learn and grow in a field I deeply admire. Participating in this challenge has opened many doors and given me the chance to reflect on the types of projects I'd like to pursue in the future. Thank you for the opportunity—it was an experience I thoroughly enjoyed.

Hugging Face model link:

https://huggingface.co/Venky0404/llava-v1.6-version2-mod-prompt